M2td: Multi-task tensor decomposition for sparse ensemble simulations

M2TD: Multi-Task Tensor Decomposition for Sparse Ensemble Simulations

Xinsheng Li
Arizona State University
Tempe, AZ 85287, USA
Email: lxinshen@asu.edu

K. Selçuk Candan
Arizona State University
Tempe, AZ 85287, USA
Email:candan@asu.edu

Maria Luisa Sapino
University of Torino
I-10149 Torino, Italy
Email:marialuisa.sapino@unito.it

***Abstract*—Data- and model-driven computer simulations are increasingly critical in many application domains. These simulations may track 10s or 100s of parameters, affected by complex inter-dependent dynamic processes. Moreover, decision makers usually need to run large simulation ensembles, containing 1000s of simulations. In this paper, we rely on a tensor-based framework to represent and analyze patterns in large simulation ensemble data sets to obtain a high-level understanding of the dynamic processes implied by a given ensemble of simulations. We, further, note that the inherent sparsity of the simulation ensembles (relative to the space of potential simulations one can run) constitutes a significant problem in discovering these underlying patterns. To address this challenge, we propose a partition-stitch sampling scheme, which divides the parameter space into sub-spaces to collect several lower modal ensembles, and complement this with a novel Multi-Task Tensor Decomposition (M2TD), technique which helps effectively and efficiently stitch these sub-ensembles back. Experiments showed that, for a given budget of simulations, the proposed structured sampling scheme leads to significantly better overall accuracy relative to traditional sampling approaches, even when the user does not have perfect information to help guide the structured partitioning process.**

I. Introduction

Data- and model-driven computer simulations are increasingly critical in many application domains. For example, for predicting geo-temporal evolution of epidemics and assessing the impact of interventions, experts often rely on epidemic spread simulation software, such as STEM [6]. Simulation-based decision making, however, introduces several fundamental data challenges [23], [28]:

- Many complex processes (such as disasters [4]) involve various distinct, yet inter-dependent, sub-processes. Consequently, in order to be useful, these simulations may track 100s of parameters, spanning multiple layers and spatial-temporal frames, affected by complex inter-dependent dynamic processes (Figure 1).
- Moreover, due to large number of unknowns, decision makers usually need to generate an ensemble of stochastic realizations, requiring 1000s of individual simulation instances, each with different parameter settings corresponding to different, but plausible, scenarios.

Research is supported by NSF#1318788 "Data Management for Real-Time Data Driven Epidemic Spread Simulations", NSF#1339835 "E-SDMS: Energy Simulation Data Management System Software", NSF#1610282 "DataStorm: A Data Enabled System for End-to-End Disaster Planning and Response", NSF#1633381 "BIGDATA: Discovering Context-Sensitive Impact in Complex Systems", and "FourCmodeling": EU-H2020 Marie Sklodowska-Curie grant agreement No 690817.

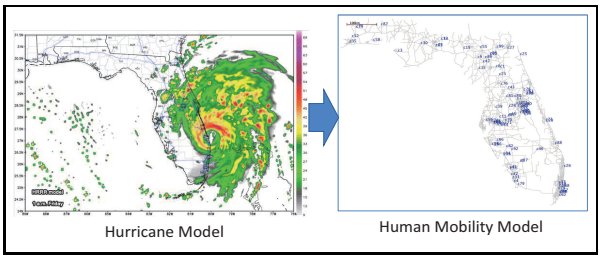


Fig. 1: Coupled simulation of a hurricane and human mobility

Consequently, obtaining and interpreting simulation ensembles to generate actionable results present difficulties:

- **Limited ensemble simulation budgets:** Since complex, inter-dependent parameters affected by complex dynamic processes have to be taken into account, *execution of simulation ensembles can be very costly*. This leads to *simulation budget constraints* that limit the number of simulations one can include in an ensemble.
- **Need for post-simulation data processing:** Because of the complexities of key processes and the varying scales at which they operate, experts often lack the means to drive conclusions from these ensembles. This leads to the need for *data analytics on simulation ensembles to discover broad, actionable patterns*.
- **Inherent data sparsity of simulation ensembles:** While the size and complexity of a simulation ensemble can indeed tax decision makers, we note that *a simulation ensemble is inherently sparse (relative to the space of potential simulations one could run)*, which constitutes a significant problem in simulation-based decision making. This leads to the following critical question: "*Given a parameter space and a fixed simulation budget, which simulation instances we should include in the ensemble?*"

A. Tensor Representation of Simulation Ensembles

In this paper, we propose a tensor-based framework to represent and analyze large simulation ensembles. Intuitively, the tensor model maps a multi-attribute schema to a multi-modal array (where each potential tuple is a tensor cell). Consequently, we can map a given simulation ensemble onto

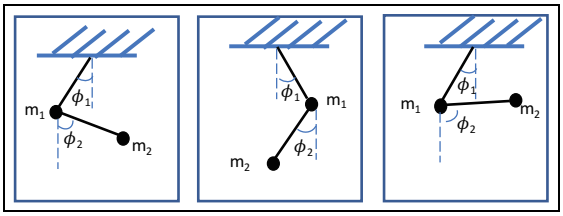


Fig. 2: States of a multi-pendulum system

a tensor such that each simulation parameter corresponds to a mode of a tensor and the non-null entries in the tensor represent results of the simulations we have executed.

Tensor decomposition [11], [32], [19] (which generalizes matrix decomposition to tensors) has been successfully used in various applications, such as social networks, sensor streams, and others [20]. Intuitively, the tensor decomposition process rewrites the given tensor in the form of a set of *factor* matrices (one for each mode of the input tensor) and a core matrix (which, intuitively, describes the spectral structure of the given tensor). As such, tensor decomposition has also been used for the analysis of dynamical systems: [29] proposed a tensor-based model for time series and [18] proposed a dynamic mode decomposition (DMD) scheme for the analysis of the behavior of complex dynamical systems.

B. Inherent Sparsity of Ensembles

While, as discussed above, tensors have been successfully used for understanding dynamic systems, we note that when the data is sparse, tensor decomposition is less effective in extracting meaningful information – which is a significant challenge when we are attempting to learn about dynamic processes through an *inherently sparse* ensemble of simulations. To see why, note that as the number of input parameters of a simulation increases, the number of potential situations one can simulate increases exponentially. Consider for example, the simple dynamical system, *double equal-length pendulum*, depicted in Figure 2: in this system there are five parameters that one can control: (a) the initial angle of the first pendulum ϕ_1, (b) the initial angle of the second pendulum ϕ_2, (c) the weight of the first bob m_1, (d) the weight of the second bob m_2, and (b) the gravity, g. For each combination of parameter values, the system can be viewed as a two-variate time series consisting of the angles of the pendulums at each time step. It is easy to see that the number of potential simulations of this *double equal-length pendulum* system is a function of the resolution of each of these four parameters – if we simply assume that for each parameter we consider, say, 20 distinct values, this would lead to $20^5 = 3200000$ possible simulations to potentially consider. Assuming that we have a simulation budget, $B = 1000$, this would lead to a simulation density of only $1000/3200000 \sim 0.0003125$. Therefore, even for a relatively small number of parameters, any realistic simulation budget is likely to be much smaller than the possible space of all simulations – consequently, the naive approach of randomly sampling the simulation space is likely to lead to sparse tensors that are difficult to accurately analyze.

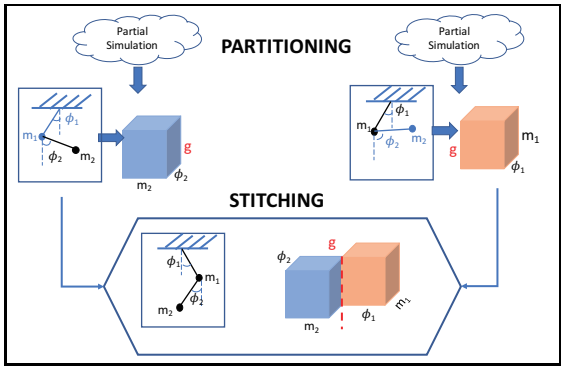


Fig. 3: Partition-Stitch sampling

C. Contribution 1: Density Boosting Partition-Stitch Sampling

In this paper, we propose an alternative ensemble creation strategy, which we refer to as the *partition-stitch sampling* (Figure 3): given an N-parameter simulation and an ensemble budget of B, instead of randomly allocating the B samples in the N-dimensional parameter space, we partition the simulation space into $\sim N/2$ dimensional sub-spaces and allocate $B/2$ simulations for each sub-space: note that, since the number of possible simulations for each sub-space reduced exponentially (in the number of excluded parameters), this corresponds to an exponential increase in the density of the samples for each sub-space: let us re-consider the *double equal-length pendulum* system in Figure 2: instead of considering the original 5-parameter system, we can divide the simulation space into simulations for two 3-parameter systems:

- **System 1:** In this system, we are allowed to *vary* the initial angle, ϕ_1, and weight, m_1, of the first pendulum as well as the gravity, g; but the initial angle, ϕ_2, and weight, m_2, of the second pendulum are *fixed*.
- **System 2:** In the second system, we can *vary* the initial angle, ϕ_2, and weight, m_2, of the second pendulum as well as the gravity, g; in this case, the initial angle, ϕ_1, and weight, m_1, of the first pendulum are *fixed*.

Note that neither of the two systems are perfect representations of the overall behavior of the whole system as, in both cases, two out of the five parameters are fixed to some default values. However, the simulation densities of both systems are now much higher than the simulation density of the original system: using the numbers considered earlier, each sub-system has 3 parameters with 20 distinct values, leading to a parameter space of $20^3 = 8000$ simulations. If we allocate 500 (=1000/2) simulations to each sub-space, this leads to a simulation density of $500/8000 = 0.0625$, which is 200 time denser than the original simulation space. There, however, remain several important questions:

- The first important question is "*How do we stitch back the results obtained from the individual sub-spaces?*"

Here we may have several alternatives: In the simplest alternative, all the simulations from the two systems can be unioned into a single 5-mode tensor and this 5-mode tensor can be decomposed for analysis. This is potentially very expensive as

the decomposition cost often increases exponentially with the number of modes of the input tensor [17], [22]. We will also see that, once unioned into a single tensor, the overall density is still low and the accuracy gains will be very limited.

Instead, we will present a join-based scheme to increase the *effective density* of the ensemble. In particular, we will present two approaches (join stitching and zero-join stitching) to combine simulation results form the sub-systems and experimentally validate the effectiveness of these schemes. Several questions, however, remain:

- *How do we select the parameter to be shared across the two sub-spaces?*: We experimentally verify that the significant gains in accuracy due to the increase in simulation densities of the sub-systems reduces the need to be particularly careful in selecting the shared parameter.
- *What about the fact that both partial systems use some* default *values to fix some of the parameters? Doesn't this negatively affect accuracy?* We will see that the gains obtained in accuracy due to the significant jump in simulation densities will overcome any disadvantages associated with fixing some of the parameters.
- *If we are joining the sub-ensembles back to the original N-parameter space, wouldn't this negatively effect the tensor decomposition cost?* If done naively, yes; and we discuss this in the next sub-section.

D. Contribution 2: Multi-Task Tensor Decomposition (M2TD)

Naively joining the sub-ensembles would map the simulations back to an N-modal tensor and this would exponentially increase the tensor decomposition time. Instead, in this paper, we propose a novel **Multi-Task Tensor Decomposition** (`M2TD`) scheme, which reduces the computational complexity of high-order tensor decomposition by (a) first cheaply decomposing the low-order partial tensors and (b) intelligently stitching back the decompositions of these partial tensors to obtain the decomposition for the whole system. Intuitively, M2TD leverages partial and imperfect simulation-based knowledge from the resulting partial dynamical systems to obtain a global view of the complex process being simulated. In this paper, we study alternative ways one can stitch the tensor decompositions and propose an $\texttt{M2TD} - \texttt{SELECT}$ that provides better accuracy than the alternatives.

E. Organization of the Paper

This paper is organized as follows: In the next section, we present the related work. Section III presents the relevant notations and the background. Section IV presents several conventional solutions to the problem and outlines their weaknesses. Section V describes the proposed partition-stitch sampling technique supported with a novel multi-task tensor decomposition approach (Section VI). Section VII experimentally evaluates the effectiveness of the `M2TD` and its alternative implementations. Experiments show that `M2TD` indeed improves the decomposition accuracy of high order tensors and handles much larger datasets than the current state of the art. We conclude the paper in Section VIII.

II. Related Work

A. Simulation Design

Ensemble simulations are increasingly critical in many application domains [28], [30]. Yet, (a) designing an ensemble that appropriately covers the input parameter space [9] and (b) interpreting the simulations in the ensemble [23], [28] are not trivial.

Work in this area, primarily focused on the first problem, which is often handled through single- or multiple-run replications [26]: in single-run replication, simulation instances are allocated incrementally, at each step evaluating the performance and deciding the next simulation to run; in multiple-run replication, the parameter space is sampled simultaneously, resulting in multiple-shorter runs. A long line of work in the area is, then, focused on the development of performance estimators and experiment design strategies [10], [15]. More recently, budget constraints and costs of simulations are being taken into account in simulation instance selection [25]. In this paper, however, we note that the second problem is as important as the first one, and therefore one has to consider the two problems of designing ensemble simulations under budget constraints and interpreting the results – we therefore, propose, a tensor-based framework for ensemble simulations and present a partition-stitch strategy to effectively increase the ensemble density to provide more accurate tensor-based analysis of a given ensemble.

B. Tensors and Scalable Tensor Decomposition

As discussed earlier, tensor decomposition (such as CP [11] and Tucker [32]) is commonly used for analyzing multi-dimensional data [20]. Yet, the tensor decomposition process is subject to several major challenges: decomposition algorithms have high computational costs and, in particular, incur large memory overheads (also known as the *intermediary data blow-up problem*) and, thus, basic algorithms and naive implementations are not suitable for large problems.

There are two widely used toolboxes for tensor manipulation: the *Tensor Toolbox for Matlab* [8] (for sparse tensors) and *N-way Toolbox for Matlab*[7] (for dense tensors). Parallel implementations, such as GridParafac [27], GigaTensor [16], HaTen2 [14], TensorDB [17], [22], [21], were proposed to deal with the high computational cost of the task. [31] proposes MACH, a randomized algorithm that speeds up the Tucker decomposition while providing accuracy guarantees. In [24], authors propose PARCUBE, a sampling based, parallel and sparsity promoting, approximate PARAFAC decomposition scheme. Scalability is achieved through sketching of the tensor (using biased sampling) and parallelization of the decomposition operations onto the resulting sketches. TensorDB [17], [22], [12] leverages a block-based framework to store and retrieve data, extends array operations to tensor operation, and introduces optimization schemes for in-database tensor decomposition. HaTen2 [14] focuses on sparse tensors and presents a scalable tensor decomposition suite on a MapReduce framework. SCOUT [13] is a recent coupled matrix-tensor factorization framework, also built on MapReduce.

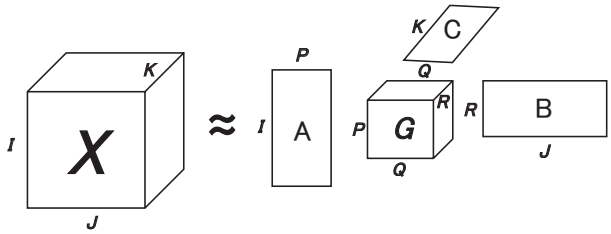


Fig. 4: Tucker decomposition of a three-mode tensor

III. Background and Notations

A. Tensors

The tensor model maps a multi-attribute schema into an N-modal array. More formally, let I^j denote the number of distinct values that the j^{th} attribute (or j^{th} mode) can take. The tensor $\boldsymbol{\mathcal{X}}$ is then an N mode array such that $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$. Intuitively, the modes of the tensor represent different *factors* that impact an observation and the value that the tensor records for a given cell corresponds to an observation for a specific combination of factor instances.

B. Tensor Decomposition

The two most popular tensor decomposition algorithms are the Tucker [32] and the CANDECOMP/PARAFAC (CP) [11] decompositions. In this paper, we focus on the Tucker decomposition of simulation ensemble tensors. Intuitively, the Tucker decomposition generalizes singular value matrix decomposition (SVD) to higher-dimensional data (Figure 4). Given a tensor $\boldsymbol{\mathcal{X}}$, Tucker decomposition factorizes the tensor into factor matrices with different number of rows, which are referred to as the rank of the decomposition. For the simplicity of the discussion, let us consider a 3-mode tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$. Tucker decomposition would decompose $\boldsymbol{\mathcal{X}}$ into three matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and one core dense tensor $\mathbf{g}$, such that

$$\boldsymbol{\mathcal{X}} \approx \tilde{\boldsymbol{\mathcal{X}}} = \mathbf{g} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} \equiv \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} a_p \circ b_q \circ c_r,$$

where $\mathbf{A} \in \mathbb{R}^{I \times P}$, $\mathbf{B} \in \mathbb{R}^{J \times Q}$, $\mathbf{C} \in \mathbb{R}^{K \times R}$, are the factor matrices and can be treated as the principal components in each mode. The (dense) core tensor, $\mathbf{g} \in \mathbb{R}^{P \times Q \times R}$, indicates the strength of interactions among different components of the factor matrices.

It is important to note that tensors very rarely have exact Tucker decompositions. In almost all cases, the new tensor $\tilde{\boldsymbol{\mathcal{X}}}$ obtained by recomposing the factor matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and core tensor $\mathbf{g}$ is often different from the input tensor, $\boldsymbol{\mathcal{X}}$. The accuracy of the decomposition is often measured by considering the Frobenius norm of the difference tensor.

More generally, given an N-mode tensor, $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, and N target rank values, r_1 through r_N, the corresponding Tucker decomposition is $[\boldsymbol{\mathcal{G}}, \mathbf{U^{(1)}}, \mathbf{U^{(2)}}, \mathbf{U^{(3)}}, \ldots, \mathbf{U^{(N)}}]$, such that

$$\tilde{\boldsymbol{\mathcal{X}}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{U^{(1)}} \times_2 \mathbf{U^{(2)}} \times_3 \mathbf{U^{(3)}} \ldots \times_N \mathbf{U^{(N)}} \approx \boldsymbol{\mathcal{X}}.$$

Here, $\mathbf{U^{(i)}}$ are the N factor matrices and $\boldsymbol{\mathcal{G}}$ is an $r_1 \times \ldots \times r_N$ dimensional core tensor.

Algorithm 1 HOSVD

Input: Tensor $\boldsymbol{\mathcal{X}}$, Rank for each mode $r_1, r_2, ..., r_N$
Output: Decomposed factors $\boldsymbol{U}^{(1)}$, $\boldsymbol{U}^{(2)}$, ..., $\boldsymbol{U}^{(N)}$ and core tensor $\boldsymbol{\mathcal{G}}$
for $n = 1, ..., N$ **do**
  matricize $\boldsymbol{\mathcal{X}}$ into matrix $X_{(n)}$
  $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{(n)}$
end
$\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{X}} \times_1 U^{(1)T} \times_2 U^{(2)T}, ..., \times_N U^{(N)T}$
return $\boldsymbol{\mathcal{G}}$, $U^{(1)}$, $U^{(2)}$,..., $U^{(N)}$

1) HOSVD Decomposition [20] : Algorithm 1 illustrates the HOSVD algorithm for Tucker decomposition of a given N-mode tensor, $\boldsymbol{\mathcal{X}}$ for target rank values, r_1 through r_N. For each mode of the tensor, HOSVD matricizes (flattens) the high-order tensor $\boldsymbol{\mathcal{X}}$ into a matrix. Then this matrix is decomposed (using SVD) to obtain the left eigenvectors and these are packed into a factor matrix for the corresponding mode. Finally, the core tensor is recovered from the original tensor and the N factor matrices obtained as described.

C. Tensor Representation of a Complex System

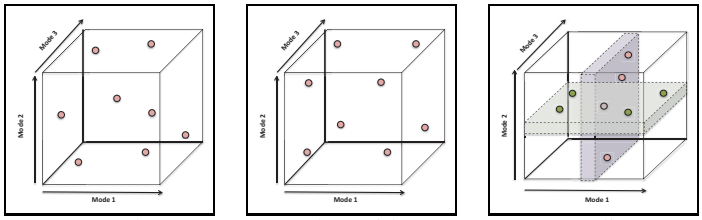
Let us be given a complex dynamic system, S, with N input parameters, such that the i^{th} input parameter can take I_i distinct values. For simplicity of the discussion, let us further assume that for each input parameter combination $\langle v_1, \ldots v_N \rangle$, the complex dynamic system S generates a single value $S(v_1, \ldots, v_n)$. Let, further, Y be the set of all simulations of the system S one can execute and the corresponding results; i.e., $Y = \{y_i = \langle \langle v_{i,1}, \ldots, v_{i,N} \rangle, S(v_{i,1}, \ldots, v_{i,N}) \rangle \parallel 1 \leq i \leq I_1 \times I_2 \times \ldots \times I_N\}$. It is easy to see that Y can be encoded as a tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, where for all $y_i \in Y$, the tensor cell $\boldsymbol{\mathcal{Y}}(v_{i,1}, \ldots, v_{i,N})$ has the value $S(v_{i,1}, \ldots, v_{i,n})$.

D. Tensor Representation of a Simulation Ensemble

The number, $I_1 \times \ldots \times I_N$, of simulations of the system, S, one can run can be very large. Instead, as discussed in the introduction, we often run a much smaller subset (or ensemble) of the simulations to get an idea about S. Given an ensemble of $B \ll I_2 \times \ldots \times I_N$ simulations, let X be the set of simulations that have been selected to be executed as well as the corresponding system outputs; i.e., $X = \{x_i = \langle \langle v_{i,1}, \ldots, v_{i,N} \rangle, S(v_{i,1}, \ldots, v_{i,N}) \rangle \parallel 1 \leq i \leq B\}$. It is easy to see that X can be encoded as a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, where for all $x_i \in X$, the tensor cell $\boldsymbol{\mathcal{X}}(v_{i,1}, \ldots, v_{i,N})$ has the value $S(v_{i,1}, \ldots, v_{i,N})$ and all other cells have null values (indicating simulations that could potentially have been run, but have not been included in the ensemble). Since $B \ll I_1 \times I_2 \times \ldots \times I_N$, the tensor $\boldsymbol{\mathcal{X}}$ is very sparse, meaning that there will be many more null-valued cells than the cells recording real-valued simulation results.

E. Problem Definition

Ideally, to study the system, S, we would construct a complete tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$, and given target rank values,



(a) Random (b) Grid (c) Slice

Fig. 5: Conventional solutions for ensemble generation

r_1 through r_N, we would obtain its corresponding Tucker decomposition $[\boldsymbol{\mathcal{H}}, \mathbf{V^{(1)}}, \mathbf{V^{(2)}}, \mathbf{V^{(3)}}, \ldots, \mathbf{V^{(N)}}]$, where

$$\tilde{\boldsymbol{\mathcal{Y}}} = \boldsymbol{\mathcal{H}} \times_1 \mathbf{V^{(1)}} \times_2 \mathbf{V^{(2)}} \times_3 \mathbf{V^{(3)}} \ldots \times_N \mathbf{V^{(N)}} \approx \boldsymbol{\mathcal{Y}}.$$

However, this would be prohibitively costly:

- Firstly, this would require $I_1 \times \ldots \times I_N$ simulations, which can be computationally overwhelming.
- Even if this many simulations can be obtained, the analysis of the resulting tensor may be prohibitively expensive.

Instead, given a budget $B \ll I_1 \times \ldots \times I_N$ of simulations, the problem is to identify a set, $X = \{x_i = \langle\langle v_{i,1}, \ldots, v_{i,N}\rangle, S(v_{i,1}, \ldots, v_{i,N})\rangle \parallel 1 \leq i \leq B\}$ of B simulations to execute, such that the Tucker decomposition $[\boldsymbol{\mathcal{G}}, \mathbf{U^{(1)}}, \mathbf{U^{(2)}}, \mathbf{U^{(3)}}, \ldots \mathbf{U^{(N)}}]$ of the corresponding tensor $\boldsymbol{\mathcal{X}}$ has the following property:

$$\tilde{\boldsymbol{\mathcal{X}}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{U^{(1)}} \times_2 \mathbf{U^{(2)}} \times_3 \mathbf{U^{(3)}} \ldots \times_N \mathbf{U^{(N)}} \approx \boldsymbol{\mathcal{X}},$$

and the Frobenius norm, $\|\boldsymbol{\mathcal{Y}} - \tilde{\boldsymbol{\mathcal{X}}}\|_F$, of the difference (from the full simulation ensemble, $\boldsymbol{\mathcal{Y}}$) is small.

IV. Conventional Ensemble Sampling Strategies

A. Strategy #1: Random Sampling

The first approach for creating a budget constrained ensemble of simulations for the system S is to uniformly randomly sample $B \ll I_1 \times \ldots \times I_N$ parameter value configurations in the parameter space and execute those B randomly sampled simulations to obtain the ensemble, X_{rs} (Figure 5(a)).

B. Strategy #2: Grid Sampling

The second approach for creating a budgeted ensemble of simulations for S is to sample B parameter value configurations at positions defined by a regularly spaced grid and execute those B sampled simulations to obtain the ensemble, X_{gs} (Figure 5(b)).

C. Strategy #3: Slice Sampling

As we can see from Figures 5(a) and (b), the major difference between random sampling and grid sampling is that in grid-based ensemble construction, the subsets of the selected simulation samples are aligned on vertical and horizontal directions (or slices) of the underlying tensor and these vertical and horizontal slices cover the tensor regularly. Alternatively, these slices and the samples within each slice can be randomly selected. Intuitively, each slice fixes one of the parameters, therefore, the samples within each slice are denser (whereas the density of the overall tensor remains the same). We refer to the resulting ensemble as X_{ss}.

V. Partition-Stitch Sampling

The three alternatives presented in the previous section cover the underlying parameter space in different ways using the same number of simulation instances. Consequently, while the local sub-space densities may differ, the overall simulation density is identically low for all three cases.

In this section, we show that, while executing the same number (B) of simulation instances as before, we can *increase the effective simulation density* of the ensemble by carefully partitioning the simulations to run into two groups and, then, by carefully stitching them, relying on *shared information* among these groups to *transfer* knowledge among them.

A. Key Observation

The key observation is that most complex processes can be partitioned such that, while each partition captures different sub-processes, these nevertheless relate to each other and, hence, reflect the footprints of the same underlying global pattern. Therefore, at least in theory, it should be possible to partition the given system S into two sub-systems S_1 and S_2, and analyze them independently. *Transferring* what we independently learned from the analysis of S_1 and S_2 back-and-forth, we should be able to gather information regarding the original global system, S. To leverage this observation, however, we need to answer two major questions: (a) "*How do we partition the system, S, into two sub-systems?*" and (b) "*How do we stitch the outcomes of these two sub-systems, S_1 and S_2, back to learn about S?*"

B. PF-Partitioning of a Parameter Space

It turns out that the answer to the first question is relatively straightforward: Given a system S with N input parameters, we will partition the system into two sub-systems S_1 and S_2, each with $\frac{N-k}{2} + k$ input parameters, such that

- the two systems share k of their input parameters as *pivot parameters*, and
- for each system, the remaining $\frac{N-k}{2}$ parameters will be set to a default value, referred to as *fixing constants*.

We will refer to this as the *Pivoted/Fixed (PF)-partitioning* of a parameter space. Intuitively, S_1 and S_2 correspond to two *constrained sub-spaces*: they have lesser free parameters than the original system S as each one is generated by fixing $\frac{N-k}{2}$ of the input parameters. Once the two sub-systems are obtained through PF-partitioning, we can then create two sets, X_1 and X_2, of ensembles (through random, grid, or slice sampling), each with $B/2$ simulations – these simulations are created with common values for shared pivot parameters. Consequently, the pivot parameters can be used for stitching the two ensembles together. More formally, let $\rho_1, \ldots, \rho_i, \ldots \rho_N$ denote the N input parameters of S, each with a domain with I_i distinct values. Without loss of generality, we refer to

- ρ_1 through ρ_k as the pivot parameters,
  - we select $P \leq I_1 \times \ldots \times I_k$ possible configurations for the pivot parameters for ensemble generation,

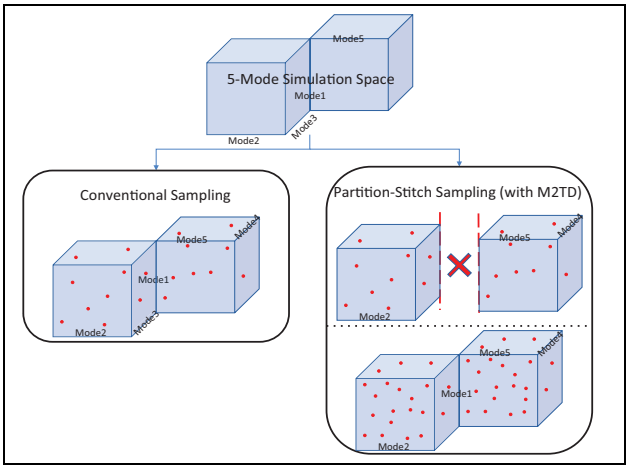


Fig. 6: Ensemble creation through PF-partitioning, followed by JE-stitching provides a higher effective density than the convention sampling of the original parameter space

- ρ_{k+1} through $\rho_{k+(N-k)/2}$ will serve as the free input parameters of system S_1 and fixed parameters of S_2,
  - we select $E \leq I_{k+1} \times \ldots \times I_{k+(N-k)/2}$ possible configurations for the free parameters for ensemble generation for system S_1,
- $\rho_{k+(N-k)/2+1}$ through ρ_N will serve as the free input parameters of system S_2 and fixed parameters of S_1.
  - we select $E \leq I_{k+(N-k)/2+1} \times \ldots \times I_N$ possible configurations for the free parameters for ensemble generation for system S_2.

Note that, given the input budget B, we have $P \times E = B/2$. In the next sub-section, we discuss how to stitch these sub-ensembles to increase the overall effective density.

C. JE-Stitching

As we mentioned above, the goal of the stitching process is to increase the effective density of the ensemble. *Join-Ensemble (JE)-Stitching* achieves this by *joining* or *zero-joining* the two sub-systems along the shared modes:

1) Join-based Stitching: Let $\mathcal{X}_1$ and $\mathcal{X}_2$ denote the two tensors representing the simulation ensembles, X_1 and X_2, for the two sub-systems $S_1(\rho_{1,1}, \ldots, \rho_{1,k+(N-k)/2})$ and $S_2(\rho_{2,1}, \ldots, \rho_{2,k+(N-k)/2})$, respectively. For simplicity, let the first k parameters of both sub-systems denote the set of parameters shared between the two sub-systems. We construct a new join ensemble, J, as follows: for all pairs of simulations in the two ensembles that agree on the parameter values for the k shared parameters (i.e., $(\rho_{1,1} = \rho_{2,1}) \wedge \ldots \wedge (\rho_{1,k} = \rho_{2,k})$), we compute the average of the terms

$$x_1 = X_1(\rho_{1,1}, \ldots, \rho_{1,k}, \rho_{1,k+1}, \ldots, \rho_{1,k+(N-k)/2})$$

$$x_2 = X_2(\rho_{2,1}, \ldots, \rho_{2,k}, \rho_{2,k+1}, \ldots, \rho_{2,k+(N-k)/2})$$

and the resulting average, $\frac{x_1+x_2}{2}$, as the value for the corresponding join ensemble entry $J(\rho_{1,1}, \ldots, \rho_{1,k+(N-k)/2}, \rho_{2,k+1}, \ldots, \rho_{2,k+(N-k)/2})$.

Note that, since for each one of the P unique combinations selected for the shared pivot parameters, there are E ensemble simulations in both sub-systems, the resulting join ensemble tensor, $\mathcal{J}$, represents $P \times E^2$ joined simulations – since, as we saw in the previous subsection, we have $P \times E = B/2$, this gives us $B^2/(4P)$ simulation entries, (and assuming that $B \gg (4P)$) *effectively squaring the simulation density* (Figure 6). As we experimentally verify in Section VII, (due to this increased effective density) the decomposition of $\mathcal{J}$ will be a far better approximation for the original system S then the decomposition of the tensor $\mathcal{X}$ which represents the original set of simulations, $X = X_1 \cup X_2$. In fact, the accuracy gains associated with this density increase

- prevents any disadvantages associated with eliminating some of the free parameters, and
- leads to significant overall accuracy gains, even without precise *a priori* knowledge about parameters to use as pivot and/or values for *fixing constants*.

2) Zero-Join based Stitching: Note, however, that when E (i.e., sub-system densities) is small, the overall join ensemble density may still be too low to provide accurate analysis. In such a case, we can further boost the overall ensemble density by using *zero-join* (as opposed to simple join) to stitch the sub-ensembles: when constructing the join ensemble, J, for all pairs of simulations in the two sub-ensembles that agree on the parameter values for the k shared parameters (i.e., $(\rho_{1,1} = \rho_{2,1}) \wedge \ldots \wedge (\rho_{1,k} = \rho_{2,k})$), we still compute the average of the terms as described above. But, in this case, if there is a simulation instance,

$$x_1 = X_1(\rho_{1,1}, \ldots, \rho_{1,k}, \rho_{1,k+1}, \ldots, \rho_{1,k+(N-k)/2})$$

but the simulation instance

$$X_2(\rho_{1,1}, \ldots, \rho_{1,k}, \rho_{2,k+1}, \ldots, \rho_{2,k+(N-k)/2})$$

does not exist; then we treat the *missing* simulation instance as if it exists with 0 value, and we construct the corresponding join ensemble entry $J(\rho_{1,1}, \ldots, \rho_{1,k+(N-k)/2}, \rho_{2,k+1}, \ldots, \rho_{2,k+(N-k)/2})$ with value $\frac{x_1+0}{2}$. We similarly handle simulation instances in X_2.

Note that zero-joining increases the effective density of the simulation ensemble to $2 \times (P \times E^2) \times E^2$, and as we experimentally verify in Section VII, it significantly boosts accuracy in cases where sub-ensemble simulation densities are too low for basic join-based stitching be effective.

VI. Multi-Task Tensor Decomposition (M2TD)

The difficulty with JE-stitching, of course, is that tensor $\mathcal{J}$ has almost double the number of modes as the tensors $\mathcal{X}_1$ and $\mathcal{X}_2$. Consequently, its decomposition is likely to be significantly more expensive than the decomposition of these two pre-join tensors. What remains to be shown is that we can, in fact, obtain the decomposition of $\mathcal{J}$ directly from the decompositions of $\mathcal{X}_1$ and $\mathcal{X}_2$. We discuss this in this section.

Let $\mathcal{X}_1$ and $\mathcal{X}_2$ be two sub-ensemble tensors corresponding to sub-systems constructed through PF-partitioning of an N-parameter system, S. Let J be the join ensemble and $\mathcal{J}$

Algorithm 2 M2TD-AVG

Input: Tensors $\mathcal{X}_1$ and $\mathcal{X}_2$, Rank for each mode $r_1, r_2, ..., r_N$
Output: Decomposed factors $U^{(1)}$, $U^{(2)}$, ..., $U^{(N)}$ and core tensor $\mathcal{G}$ for the join tensor $\mathcal{J}$

for $m = 1, ..., M$ **do**
  matricize $\mathcal{X}_1$ into matrix $X_{1(m)}$
  matricize $\mathcal{X}_2$ into matrix $X_{2(m)}$
end
for $n = 1, ..., k$ **do**
  $U^{1(n)} \leftarrow r_n$ leading left singular vectors of $X_{1(n)}$
  $U^{2(n)} \leftarrow r_n$ leading left singular vectors of $X_{2(n)}$
  $U^{(n)} \leftarrow average(U^{1(n)}, U^{2(n)})$
end
for $n = k+1, ..., M$ **do**
  $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{1(n)}$
end
for $n = M+1, ..., 2M-k$ **do**
  $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{2(n-M+k)}$
end
$\mathcal{J}$ = join_tensor($\mathcal{X}_1$,$\mathcal{X}_2$)
$\mathcal{G} = \mathcal{J} \times_1 U^{(1)T} \times_2 U^{(2)T}, ..., \times_N U^{(N)T}$
return $\mathcal{G}$, $U^{(1)}$, $U^{(2)}$,..., $U^{(N)}$

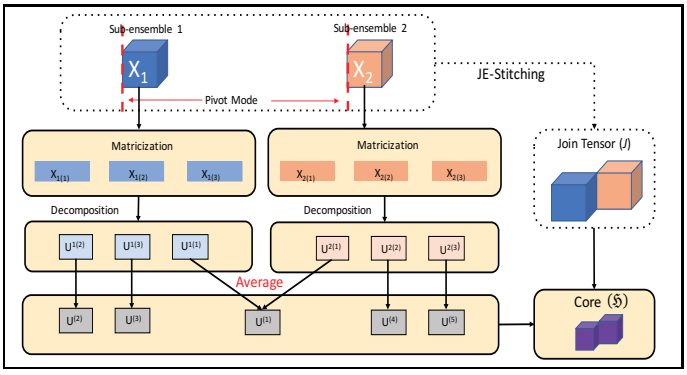


Fig. 7: Overview of M2TD-AVG

Algorithm 3 M2TD-CONCAT

Input: Tensors $\mathcal{X}_1$ and $\mathcal{X}_2$, Rank for each mode $r_1, r_2, ..., r_N$
Output: Decomposed factors $U^{(1)}$, $U^{(2)}$, ..., $U^{(N)}$ and core tensor $\mathcal{G}$ for the join tensor $\mathcal{J}$

for $n = 1, ..., k$ **do**
  matricize $\mathcal{X}_1$ into matrix $X_{1(n)}$
  matricize $\mathcal{X}_2$ into matrix $X_{2(n)}$
  $X_{(n)} \leftarrow concatenate(X_{1(n)}, X_{2(n)})$
  $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{(n)}$
end
for $m = k+1, ..., M$ **do**
  matricize $\mathcal{X}_1$ into matrix $X_{1(m)}$
  matricize $\mathcal{X}_2$ into matrix $X_{2(m)}$
end
for $n = k+1, ..., M$ **do**
  $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{1(n)}$
end
for $n = M+1, ..., 2M-k$ **do**
  $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{2(n-M+k)}$
end
$\mathcal{J}$ = join_tensor($\mathcal{X}_1$,$\mathcal{X}_2$)
$\mathcal{G} = \mathcal{J} \times_1 U^{(1)T} \times_2 U^{(2)T}, ..., \times_N U^{(N)T}$
return $\mathcal{G}$, $U^{(1)}$, $U^{(2)}$,..., $U^{(N)}$

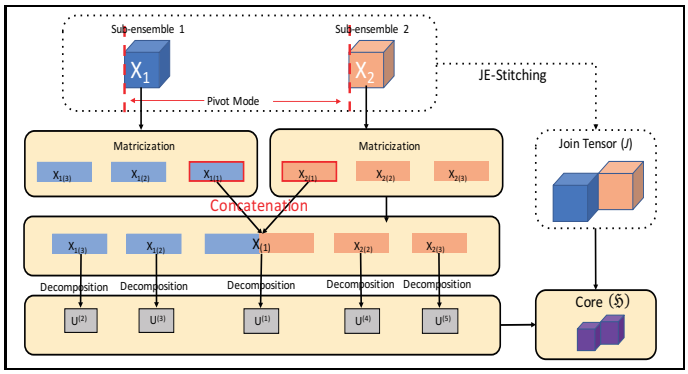


Fig. 8: Overview of M2TD-CONCAT

be the corresponding join tensor one could obtain through JE-stitching. In this section, we introduce three alternative *multi-task tensor decomposition* (M2TD) schemes to obtain the decomposition of $\mathcal{J}$ from the decompositions of $\mathcal{X}_1$ and $\mathcal{X}_2$.

A. M2TD-Average (M2TD-AVG)

Remember from the earlier sections that both $\mathcal{X}_1$ and $\mathcal{X}_2$ are M-modal tensors, where $M = k+(N-k)/2$, and that the first k modes are shared. We modify the HOSVD algorithm, presented in Section III-B1, to obtain the proposed M2TD-AVG algorithm (Algorithm 2). Intuitively, M2TD-AVG takes the first k factor matrix pairs, $(U^{1(n)}, U^{2(n)})$, corresponding to the shared pivot tensors of the independently decomposed tensors, $\mathcal{X}_1$ and $\mathcal{X}_2$, and averages each pair to obtain a *common* factor matrix representing both tensors: since factor matrices, $U^{1(n)}$ and $U^{2(n)}$, both map the domain of the corresponding factor to a vector space represented by r_n singular factors (sorted in decreasing order of significance), the operation $average(U^{1(n)}, U^{2(n)})$ essentially constructs a new vector space, where each element of the domain is represented by the average vector from the two input vector spaces (Figure 10(a)). Remaining factor matrices are then combined to obtain the core tensor, $\mathcal{G}$ (see Figure 7). As we experimentally verify in Section VII, this leads to a better approximation of the original system than any of the naive ensemble sampling schemes.

B. M2TD-Concatenate (M2TD-CONCAT)

M2TD-AVG, presented in the previous subsection, recovers the factor matrices for pivot parameters (modes) by averaging the corresponding factor matrices; i.e., by first obtaining the singular vectors of the matricizations and then averaging these singular vectors. However, there is nothing that guarantees that these averages will act as singular vectors themselves.

Instead, the alternative M2TD-CONCAT algorithm (detailed in Algorithm 3 and visualized in Figure 8) avoids this potential

Algorithm 4 M2TD-SELECT

Input: Tensors $\mathcal{X}_1$ and $\mathcal{X}_2$, Rank for each mode $r_1, r_2, ..., r_N$
Output: Decomposed factors $U^{(1)}$, $U^{(2)}$, ..., $U^{(N)}$ and core tensor $\mathcal{G}$ for the join tensor $\mathcal{J}$

for $m = 1, ..., M$ **do**
  matricize $\mathcal{X}_1$ into matrix $X_{1(m)}$
  matricize $\mathcal{X}_2$ into matrix $X_{2(m)}$
end
for $n = 1, ..., k$ **do**
  $U^{1(n)} \leftarrow r_n$ leading left singular vectors of $X_{1(n)}$
  $U^{2(n)} \leftarrow r_n$ leading left singular vectors of $X_{2(n)}$
  $U^{(n)} \leftarrow row\_select(U^{1(n)}, U^{2(n)})$
end
for $n = k+1, ..., M$ **do**
  $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{1(n)}$
end
for $n = M+1, ..., 2M-k$ **do**
  $U^{(n)} \leftarrow r_n$ leading left singular vectors of $X_{2(n-M+k)}$
end
$\mathcal{J}$ = join_tensor($\mathcal{X}_1$,$\mathcal{X}_2$)
$\mathcal{G} = \mathcal{J} \times_1 U^{(1)T} \times_2 U^{(2)T}, ..., \times_N U^{(N)T}$
return $\mathcal{G}$, $U^{(1)}$, $U^{(2)}$,..., $U^{(N)}$

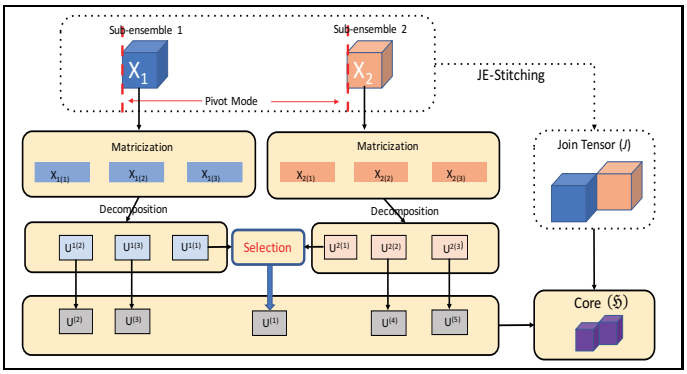


Fig. 9: Overview of M2TD-SELECT

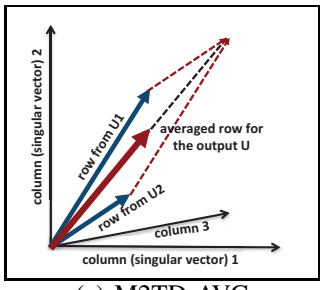
issue by first constructing a concatenated matricization for each pivot mode pair and then seeking the left singular vectors of this combined matricization. Intuitively, M2TD-CONCAT maps the matricizations along the shared/pivot modes back into the higher-modal space and seeks the singular vectors that best represent this higher modal space.

C. M2TD-Selection (M2TD-SELECT)

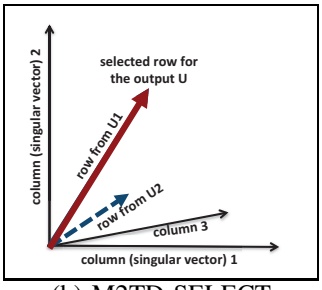
The M2TD-CONCAT algorithm presented above tries to improve the vector averaging scheme of M2TD-AVG through row-by-row concatenation of the pivot matricizations before the corresponding factor matrices are computed. In this subsection, we note that there is an alternative, and potentially more effective, way to improve the M2TD-AVG scheme: once the factor matrices for the pivots are obtained, instead of averaging them, we can carefully select between the individual rows of the corresponding factor matrices and use these selected rows to construct more effective combined factor matrices.

Algorithm 5 ROW_SELECT

Input: Factor matrices U_1 and U_2
Output: Row-selected Factor Matrix U
$I \leftarrow num\_rows(U_1)$
for $1 \leq i \leq I$ **do**
  if $\|row(U_1, i)\|_2 \geq \|row(U_2, i)\|_2$ **then**
    $row(U, i) \leftarrow row(U_1, i)$
  else
    $row(U, i) \leftarrow row(U_2, i)$
  end
end
return U



(a) M2TD-AVG



(b) M2TD-SELECT

Fig. 10: Comparison of the row construction processes between M2TD-AVG and M2TD-SELECT

The pseudocode for the process is shown in Algorithm 4 and visualized in Figure 9. Note that the major difference between this algorithm and M2TD-AVG is the line

$$U^{(n)} \leftarrow row\_select(U^{1(n)}, U^{2(n)}),$$

where the factor matrix $U^{(n)}$ is constructed by selecting the appropriate rows from $U^{1(n)}$ or $U^{2(n)}$, instead of simply averaging them. This row selection process is further detailed in Algorithm 5 and visualized in Figure 10(b). As we see here, the key idea is to consider the energies (captured by the 2-norm function) of each row, i, in U_1 and U_2, and identify which of the two factor matrices provides a higher energy for that particular row. Intuitively, this enables us to identify which of the two factor matrices better represents the entity corresponding to row, i, and, given this information, we can construct the row i of the output factor matrix, U, by selecting the corresponding row from the factor matrix, U_1 or U_2, with a higher representation power for that entity.

As we experimentally verify in Section VII, this selection strategy prevents the row with the lesser energy to act as *noise* on the description of the corresponding entity and, thus, leads to significantly higher decomposition accuracies. Moreover, as the experiments show, the accuracy gains get higher as we target higher ranking decompositions that maintain more details by seeking a larger number of patterns in the data.

D. Distributed M2TD (D-M2TD)

As discussed in Section II, a major challenge with tensor decomposition is its computational and space complexity. This is especially true for the Tucker decomposition with a dense core. In this section, relying on several key properties of the M2TD algorithm, we propose a 3-phase distributed version

Algorithm 6 The outline of the Distributed Multi-Task Tensor Decomposition, D − M2TD, process

Input: Tensor $\mathcal{X}_1$, $\mathcal{X}_2$, Rank for each mode $r_1, r_2, \ldots, r_N$
Output: Factor Matrices $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, ..., $\mathbf{U}^{(N)}$ and core tensor $\mathcal{G}$ for the join tensor $\mathcal{J}$

1) Phase 1: Parallel decomposition of $\mathcal{X}_1$ and $\mathcal{X}_2$ to generate $U^{1(n)}, U^{2(n)}, n \in \{1, \ldots, N\}$
2) Phase 2: Parallel JE-Stitching $\mathcal{X}_1$, $\mathcal{X}_2$ to obtain the decomposition of the joined tensor $\mathcal{J}$
3) Phase 3: for $1 \leq n \leq N$
   a) Parallel tensor matrix mutiplication- $\mathcal{G}_n = \mathcal{J} \times_n \mathbf{U}^{(n)}$
4) Return Factor Matrices $\mathbf{U}^{(1)}$, $\mathbf{U}^{(2)}$, ..., $\mathbf{U}^{(N)}$ and core $\mathcal{G}$

	Alternative values		
Dynamic systems	**Double Pend.**;	Triple Pend.	Lorenz System
Parameter resolution	60 ;	**70**;	80
Size of the corresponding simulation space (S)	$60^5(8 \times 10^8)$;	$70^5(2 \times 10^9)$;	$80^5(3 \times 10^9)$
Pivot density (P)	10%;	**100%**	
Sub-system density (E)	10%;	**100%**	
Ensemble budget ($B = 2 \times P \times E \times S$)	4×10^4, 4×10^5,	7×10^4, 7×10^5,	1×10^5, 1×10^6
Target decomposition rank (r)	5;	**10**;	20
Stitching technique	**Join**;	Zero-Join	
Number of servers	2, 6, 10, 14, **18**		

TABLE I: Experiment setup – default values, used unless otherwise specified, are highlighted

of M2TD that can be efficiently and scalably executed on MapReduce or Spark based platforms (see Algorithm 6):

• **Phase 1: Parallel Sub-Tensor Decomposition:** Consider the M2TD-SELECT pseudocode in Algorithm 4. Here, $\mathcal{X}_1$ and $\mathcal{X}_2$ are two sub-tensors corresponding to two sub-systems constructed through PF-partitioning. These low-order sub-tensors can be decomposed (in parallel) independently from each other. Therefore, this phase can be parallelized using, for example, the popular distributed computing framework, MapReduce, using the following `map` and `reduce` operators:

- **map**: $\langle \boldsymbol{\kappa}, \rho_1, \rho_2, \ldots, \rho_M, \boldsymbol{\mathcal{X}_\kappa}(\rho_1, \rho_2, \ldots, \rho_M) \rangle$ on $\boldsymbol{\kappa}$. Here, $\boldsymbol{\kappa}$ is the low-order tensor id; i.e., $\boldsymbol{\kappa} \in \{1, 2\}$. $\rho_1, \rho_2, \ldots, \rho_M$ together give the coordinate of a cell in the low-order tensor $\boldsymbol{\mathcal{X}_\kappa}$. Key-value pairs with the same κ are shuffled to the same reducer in the form of $\langle key : \boldsymbol{\kappa}, val : \rho_1, \rho_2, \ldots, \rho_M, \boldsymbol{\mathcal{X}_\kappa}(\rho_1, \rho_2, \ldots, \rho_M) \rangle$.
- **reduce**: $\langle key{:}\boldsymbol{\kappa}, val{:}\rho_1, \rho_2, \ldots, \rho_M, \boldsymbol{\mathcal{X}_\kappa}(\rho_1, \rho_2, \ldots, \rho_M) \rangle$. The reducer processing the key, $\boldsymbol{\kappa}$, receives the non-zero elements of sub-tensor $\boldsymbol{\mathcal{X}_\kappa}$ and decomposes it into sub-factor $\boldsymbol{U^{\kappa(n)}}$, where $\boldsymbol{n}$ is the mode id, by using SVD. Finally, reducer appropriately relabels each $\boldsymbol{U^{\kappa(n)}}$ as $\boldsymbol{U^{(n)}}$ and emits each sub-factor as an independent file, with content $\langle\ key : n, value : i, j, \boldsymbol{U^{(n)}}(i, j) \rangle$. Here, $i,\ j$ are the coordinates of sub-factor $\boldsymbol{U^{(n)}}$.

Note that this step can be further parallelized by leveraging parallel Tucker decomposition techniques, such as [24], [14].

• **Phase 2: Parallel JE-Stitching to Obtain Join Tensor, $\mathcal{J}$:** The goal of the stitching process is to increase the effective density of the ensemble. JE-stitching achieves this by joining the two sub-systems along their shared pivot modes to obtain the $\mathcal{J}$ tensor. This process can be parallelized as follows:

- **map**: $\langle \boldsymbol{\kappa}, \rho_1, \rho_2, \ldots, \rho_M, \boldsymbol{\mathcal{X}_\kappa}(\rho_1, \rho_2, \ldots, \rho_M) \rangle$. Key-value pairs with the same pivot mode index $(\rho_1, \rho_2, \ldots, \rho_k)$ are shuffled to the same reducer in the form of $\langle key{:}(\rho_1, \rho_2, \ldots, \rho_k), val{:}\rho_1, \rho_2, \ldots, \rho_M, \boldsymbol{\mathcal{X}_\kappa}(\rho_1, \rho_2, \ldots, \rho_M) \rangle$.
- **reduce:** $\langle key{:}(\rho_1, \rho_2, \ldots, \rho_k), val{:}\rho_1, \rho_2, \ldots, \rho_M, \boldsymbol{\mathcal{X}_\kappa}(\rho_1, \rho_2, \ldots, \rho_M) \rangle$. The join ensemble $\boldsymbol{\mathcal{J}}(\rho_1, \rho_2, \ldots, \rho_k, \ldots)$ is constructed for all pairs of $\boldsymbol{\mathcal{X}_\kappa}$, that agree on the parameter values for the k pivot parameters.

.

• **Phase 3: Parallel Tensor-Matrix Multiplication to Recover the Core Tensor:** As we see in Section VII, the costliest part of the M2TD algorithm is the final step where the join tensor $\mathcal{J}$ is multiplied by the transposes of the factor matrices to recover the dense core tensor. We parallelize this as follows:

- **map**: $\langle \rho_1, \rho_2, \ldots, \rho_N, \boldsymbol{\mathcal{J}}(\rho_1, \rho_2, \ldots, \rho_N) \rangle$, $\langle \boldsymbol{n}, i, j, \boldsymbol{U}^{(n)}(i, j) \rangle$. Cells of $\mathcal{J}$ (from Phase 2) with index $(\rho_1, \rho_2, \ldots, \rho_{n-1}, \rho_{n+1}, \ldots, \rho_N)$ are shuffled to the same reducer in the form of $\langle key{:}(\rho_1, \rho_2, \ldots, \rho_{n-1}, \rho_{n+1}, \ldots, \rho_N), val{:}\boldsymbol{\mathcal{J}}(\rho_1, \rho_2, \ldots, \rho_N) \rangle$
- **map**: $\langle \boldsymbol{n}, i, j, \boldsymbol{U}^{(n)}(i, j) \rangle$. Outputs of Phase 1 $\langle \boldsymbol{n}, i, j, \boldsymbol{U}^{(n)}(i, j) \rangle$ are shuffled to the same reducer based on mode id n in the form of $\langle key{:}(\rho_1, \rho_2, \ldots, \rho_{n-1}, \rho_{n+1}, \ldots, \rho_N), val{:}\boldsymbol{n}, i, j, \boldsymbol{U}^{(n)}(i, j) \rangle$
- **reduce:** The reducer takes

  $\langle key : (\rho_1, .., \rho_{n-1}, \rho_{n+1}, .., \rho_N), val : \boldsymbol{\mathcal{J}}(\rho_1, , \ldots, \rho_N) \rangle$

  and

  $\langle key : (\rho_1, .., \rho_{n-1}, \rho_{n+1}, .., \rho_N), val : \boldsymbol{n}, i, j, \boldsymbol{U}^{(n)}(i, j) \rangle$

  and performs vector-matrix multiplication to emit $\langle (\rho_1, \rho_2, \ldots, \rho_{n-1}, j, \rho_{n+1}, \ldots, \rho_N), \sum_{\rho_n=1}^{I_n} \boldsymbol{\mathcal{J}}(\rho_1, , \ldots, \rho_n, \ldots, \rho_N) * \boldsymbol{U}^{(n)}(\rho_n, j) \rangle$.

In the next section, we investigate the impact of this parallelization approach to the performance of the proposed partition-stitch sampling through M2TD decomposition.

VII. Experiments

In this section, we report results of the experiments that aim to assess the effectiveness and efficiency of the proposed partition-stitch ensemble sampling strategy and the novel *multi-task tensor decomposition (M2TD)* scheme. For these experiments, we used the Chameleon cloud platform [1]: we deployed all algorithms on 18 xxlarge instances, with 8-core vCPU, 32GB memory, 160GB disk space. Distributed versions were implemented in Java 8, over Hadoop 2.7.3. The key system parameters and their value ranges are reported in Table I and explained below.

A. Dynamic Systems

In these experiments, we consider three dynamic processes: *double pendulum*, *triple pendulum*, *lorenz system* [5]. The code for these systems was obtained from [2] and [3]. These dynamic processes are selected for their varying complexities:

The ***double pendulum*** system has four parameters: initial angle, ϕ_1, and weight, $m1$, of the first pendulum as well as the initial angle, ϕ_2, and weight, $m2$, of the second pendulum.

The ***triple pendulum (with variable friction)*** system is similar, but more complex due to the addition of a third pendulum. Moreover, the system has a different set of initial parameters: the angle ϕ_1 of the first pendulum, the initial angle ϕ_2 of the second pendulum, the initial angle ϕ_3 of the third pendulum, and the friction f of whole system. Intuitively, unlike the double pendulum system, in the triple pendulum system the friction is considered as a simulation parameter.

The ***Lorenz system*** is notable for having chaotic solutions for certain initial conditions [5]. The system has four variable parameters: the coordinate of the initial position, z, and three other system parameters, σ, β, ρ.

B. Simulation Ensembles

For the above systems, we construct 5-mode simulation ensembles. Each cell of the 5-mode ensemble simulation tensor encodes the Euclidean distance between the states of the resulting simulated system and the observed system parameters at a given time stamp, for a given quadruple of simulation parameters. Intuitively, each cell encodes the relationship between a given simulation instance to a configuration observed in the real-world.

As we see in Table I, in the experiments, the size of the simulation space varied between $60^5 \sim 8 \times 10^8$ to $80^5 \sim 3 \times 10^9$ simulation instances. In contrast, the simulation instance budgets were on the order of 10^4 to 10^5, indicating that, despite the large number of simulations included in the ensembles, the resulting ensemble tensors were very sparse (densities on the order of $\sim 10^{-4}$). Despite this sparsity, for the different configurations considered in Table I, the simulation ensemble required from 25GB to 105GB data storage.

C. Alternative Ensemble Construction Schemes

In this section, we evaluated the M2TD-AVG, -CONCAT, and -SELECT strategies and compared them against the conventional (RANDOM, GRID, and SLICE) ensemble sampling approaches (Section IV). For M2TD-based schemes, we considered the case with a single pivot parameter and, to analyze worst case behavior, we sampled the sub-systems randomly.

D. Evaluation Criteria

We compared accuracy and efficiency of alternative schemes, for different target decomposition ranks, different parameter space resolutions, and simulation budgets (see Table I). To measure accuracy, we use the Frobenius norm of the difference tensor (see Section III):

$$accuracy(\tilde{\boldsymbol{\mathcal{X}}}, \boldsymbol{\mathcal{Y}}) = 1 - \left(\frac{\|\tilde{\boldsymbol{\mathcal{X}}} - \boldsymbol{\mathcal{Y}}\|}{\|\boldsymbol{\mathcal{Y}}\|_F} \right),$$

Accuracy for Double Pendulum System							
Res.	Rank	M2TD			Random	Grid	Slice
		AVG	CONCAT	SELECT			
60	5	0.49	0.49	**0.54**	1E-8	3E-4	2E-4
	10	0.50	0.50	**0.62**	2E-7	3E-4	2E-4
	20	0.52	0.53	**0.56**	5E-6	3E-4	2E-4
70	5	0.46	0.46	**0.51**	7E-9	2E-4	2E-4
	10	0.47	0.48	**0.57**	9E-8	2E-4	2E-4
	20	0.49	0.50	**0.73**	2E-6	2E-4	2E-4
80	50	0.46	0.46	**0.50**	4E-9	1E-4	1E-4
	10	0.47	0.47	**0.49**	4E-8	1E-4	1E-4
	20	0.48	0.49	**0.59**	1E-6	2E-4	1E-4

(a) Accuracy

Decomposition Time for Double Pendulum System (sec.)							
Res.	Rank	M2TD			Random	Grid	Slice
		AVG	CONCAT	SELECT			
60	5	808	797	785	203	144	167
	10	808	819	849	234	148	186
	20	1034	929	935	348	456	258
70	5	1508	1581	1594	312	209	193
	10	1696	1645	1576	379	201	244
	20	1866	1914	1995	575	744	381
80	5	3990	3591	4907	414	227	336
	10	5232	5979	6068	514	239	410
	20	5341	5707	5439	860	883	606

(b) Time (sec.)

TABLE II: Results for double pendulum system (pivot=t, $P = 100\%$, $E = 100\%$)

Decomposition Time using Different Numbers of Servers (sec.)						
Num. Servers	M2TD-SELECT			Random	Grid	Slice
	Phase 1	Phase 2	Phase 3			
2	52	817	4167	670	420	488
6	62	383	1802	464	275	318
10	61	371	1318	415	237	280
14	65	354	1279	381	214	253
18	67	363	1118	379	201	244

TABLE III: Different number of servers (Double pendulum, resolution=70, rank = 10, pivot=t, $P = 100\%$, $E = 100\%$)

where $\tilde{\boldsymbol{\mathcal{X}}}$ is the reconstructed tensor (after sampling and decomposition), while $\boldsymbol{\mathcal{Y}}$ is the tensor corresponding to the full simulation space. We also report the decomposition times.

E. Discussions of the Results

1) General Overview: Table II focuses on the double pendulum system and compares accuracies and decomposition times for various approaches considered in this paper for the different target ranks and for different parameter resolutions. As we see in the table, the M2TD-based algorithms provide *several orders better accuracy* than the conventional approaches, with the same number of simulation instances. As expected, among the conventional schemes, the Random strategy provides the worst and the Grid strategy provides the best accuracy; however, even Grid is $\sim 1000\times$ worse than the proposed M2TD-SELECT algorithm. As also expected, among the M2TD-based algorithms, M2TD-SELECT provides the best overall accuracy: moreover, the relative performance gains of M2TD-SELECT algorithm further increases for larger decomposition ranks, indicating that as we seek more detailed patterns in the ensemble, M2TD-SELECT better captures these underlying patterns in the data.

In the Table, we also see that M2TD-based algorithms are somewhat more expensive than the conventional sampling strategies; but the gains in accuracy are several orders higher

Accuracy for Different Systems						
Dyn.System	M2TD			Random	Grid	Slice
	AVG	CONCAT	SELECT			
D.P.	0.47	0.48	**0.57**	9E-8	2E-4	2E-4
T.P.	0.25	0.25	**0.31**	6E-8	2E-4	1E-4
L.S.	0.31	0.32	**0.36**	4E-8	2E-4	1E-4

(a) Accuracy

Decomposition Time for Different Systems (sec.)						
Dyn.System	M2TD			Random	Grid	Slice
	AVG	CONCAT	SELECT			
D.P.	1696	1645	1576	379	201	244
T.P.	992	1422	1106	221	180	166
L.S.	1728	1850	1705	444	230	211

(b) Time (sec.)

TABLE IV: Results for different dynamical systems (resolution=70, rank = 10, pivot=t, $P = 100\%$, $E = 100\%$)

Accuracy for Different Ensemble Budgets (B)						
Budget	M2TD			Random	Grid	Slice
	AVG	CONCAT	SEL.			
4×10^4 (join)	3.5E-5	3.4E-5	**4.1E-5**	9E-9	2E-5	2E-6
4×10^4 (zero-join)	3.3E-3	3.2E-3	**3.9E-3**	9E-9	2E-5	2E-6
4×10^5	0.47	0.48	**0.57**	9E-8	2E-4	2E-4

(a) Accuracy

Decomposition Time for Different Ensemble Budgets (B) (sec.)						
Budget	M2TD			Random	Grid	Slice
	AVG	CONCAT	SEL.			
4×10^4 (join)	200	201	200	190	175	183
4×10^4 (zero-join)	596	598	592	190	175	183
4×10^5	1696	1645	1576	379	201	244

(b) Time (sec.)

TABLE V: Results for different ensemble budgets (Double pendulum, resolution=70, rank = 10, pivot=t; note that $B = 4 \times 10^5$ corresponds to the case where both pivot, P, and sub-systems, E, have 100% densities)

than the decomposition time overheads of M2TD-based techniques. This is because, as highlighted in Section V-C, the proposed partition-stitch technique increases the *effective density* of the join ensemble. Consequently, the increase in the decomposition is *well amortized* by the increase in the effective simulation density. In these experiments, each double pendulum simulation took roughly 0.66ms. Given this, obtaining an ensemble simulation with density $70^4 (= 70^2 \times 70^2)$ would require roughly 16000 seconds (ignoring the additional time to decompose). In contrast, the proposed M2TD based techniques are able to achieve the same *effective density* by running only 2×70^2 simulations in just 46 seconds and obtain the ensemble decomposition in an additional ~ 1600 seconds. This points to the impressive performance gains provided by the proposed multi-task tensor decomposition (M2TD) technique.

One question that remains is whether we could have joined the sub-ensembles directly into tensor $\mathcal{J}$ to decompose instead of relying on the M2TD techniques: the answer to this question is a *strong no*: for the experiments reported in Table II, with the configuration of 18 xxlarge servers, direct decomposition of the resulting dense tensor was not feasible due to memory limitations.

2) Decomposition Time Distribution: Table III presents how the decomposition time is split among the three phases of the map-reduce process described in Section VI-D. The table also shows how the execution time varies as we change the

Accuracy for Different Pivot Densities (P)						
P. Density	M2TD			Random	Grid	Slice
	AVG	CONCAT	SELECT			
10%	3.5E-2	7.6E-3	**3.6E-2**	9E-9	2E-5	2E-6
100%	0.47	0.48	**0.57**	9E-8	2E-4	2E-4

(a) Accuracy

Decomposition Time for Different Pivot Densities (P) (sec.)						
P.Density	M2TD			Random	Grid	Slice
	AVG	CONCAT	SELECT			
10%	606	597	607	190	175	183
100%	1696	1645	1576	379	201	244

(b) Time (sec.)

TABLE VI: Results for different pivot densities (Double pendulum, resolution=70, rank = 10, pivot=t, $E = 100\%$)

Accuracy for Different Sub-system Densities (E)						
E. Density	M2TD			Random	Grid	Slice
	AVG	CONCAT	SELECT			
$10\%(join)$	4E-5	4E-5	**4.5E-5**	9E-9	2E-5	2E-6
10% (zero-join)	3.4E-3	3.3E-3	**3.8E-3**	9E-9	2E-5	2E-6
100%	0.47	0.48	**0.57**	9E-8	2E-4	2E-4

(a) Accuracy

Decomposition Time for Different Sub-system Densities (E) (sec.)						
E. Density	M2TD			Random	Grid	Slice
	AVG	CONCAT	SELECT			
$10\%(join)$	207	202	201	190	175	183
10% (zero-join)	602	640	617	190	175	183
100%	1696	1645	1576	379	201	244

(b) Time (sec.)

TABLE VII: Results for different sub-system densities (Double pendulum, resolution=70, rank = 10, pivot=t, $P = 100\%$)

number of servers allocated for the decomposition process. As we see in this table, as expected, the third phase where we recover the core tensor of the decomposition is the costliest step of the process. We also see that allocating more servers indeed helps bring the cost of this phase down; however, there are diminishing returns due to data communication overheads.

3) Varying Data Sets: In Table IV, we study the accuracy and decomposition time results for different dynamic systems. As we see here, also for the triple pendulum and Lorenz systems, we observe the very same pattern: M2TD-SELECT provides the best accuracy among all alternatives, providing several orders of magnitude gain in accuracy relative to the conventional schemes.

4) Varying Budgets and Zero-Joins: In the default experiments considered above, the budget was selected such that the sub-ensembles would have a perfect density of 1.0. In the first row of Table V, we reduced the ensemble budget by taking $1/10^{th}$ of the samples we considered in the previous examples. Naturally, this results in a drop in accuracy for all approaches. However, M2TD-based schemes remain several orders better than the conventional approaches.

The table also shows that when the budgets (thus sub-ensemble densities) are low, we can boost the overall accuracy by leveraging zero-joins (introduced in Section V-C), rather than using simple joins when implementing JE-stitching.

5) Varying Pivot/Sub-Ensemble Densities: Tables VI and VII show the impact of reduced pivot and sub-ensemble densities (i.e., P and E) respectively. As we see here, the overall pattern is as before: reduction in the simulation budget

Accuracy for Different Pivot Parameters						
Pivot	M2TD			Random	Grid	Slice
	AVG	CONCAT	SELECT			
t	0.47	0.48	**0.57**	9E-8	2E-4	2E-4
ϕ_1	0.35	0.36	**0.40**			
ϕ_2	0.40	0.41	**0.56**			
m_1	0.58	0.59	**0.71**			
m_2	0.41	0.40	**0.42**			

(a) Accuracy

Decomposition Time for Different Pivot Parameters (sec.)						
Pivot	M2TD			Random	Grid	Slice
	AVG	CONCAT	SELECT			
t	1696	1645	1576	379	201	244
ϕ_1	1607	1673	1673			
ϕ_2	1694	1677	1571			
m_1	1661	1512	1697			
m_2	1556	1602	1538			

(b) Time (sec.)

TABLE VIII: Results for different pivots (Double pendulum, resolution=70, rank = 10, $P = 100\%$, $E = 100\%$; 3-mode sub-systems are created in such a way that free parameters of the same pendulum are kept in the same sub-system)

reduces the overall accuracy; however, M2TD-based schemes provide significantly higher accuracy overall.

An interesting observation, however, is that (while the total number of simulations is the same) reduction in the pivot sub-ensemble density has a significantly higher impact than the reduction in the pivot density: this is because, as discussed in Section V-C, the effective density of a stitched simulation ensemble is proportional to $P \times E^2$, and thus reductions in E have a more significant impact than reductions in P: this further confirms our initial hypothesis that maintaining sub-ensemble densities high is important for accurate characterization of the system being studied.

6) Selection of the Pivot Parameter: In Table VIII, we vary the pivot parameter[1]: as we expected, which parameter is selected as the pivot has some impact on the accuracy of the proposed partition-stitch scheme. However, whichever pivot is selected, the overall accuracy is several orders of magnitude better than that of conventional schemes, indicating that we do not need very precise information about the system being studied to decide how to partition the system.

VIII. Conclusions

In this paper, we presented a tensor-based framework to represent and analyze large simulation ensembles to support decision making in the presence of complex, dynamic systems. Noting that simulation ensembles and the corresponding tensors are often extremely sparse due to the size of the potential simulation parameter space, we proposed a partition-stitch sampling scheme to help increase the *effective density* of the simulation samples to boost accuracy. We have complemented this sampling scheme with a novel *Multi-Task Tensor Decomposition (M2TD)* to efficiently stitch these sub-ensembles in a way that leverages partial and imperfect knowledge from partial dynamical systems to effectively obtain a global view of the complex process being simulated. Experiment results showed the efficiency and the effectiveness of the proposed approach relative to more conventional techniques for constructing simulation ensembles.

[1]Due to space constraints, we omit experiments where we keep the same pivot parameter, but vary the groupings of free parameters. The results are similar to the results of pivot parameter selection.

References

[1] Chameleon Cloud. 2017.
[2] Code for double and triple pendulum systems, http://www.ita.uni-heidelberg.de/ chfeder/applets.shtml, 2017.
[3] Code for the Lorenz system, http://www.algosome.com/articles/lorenz-attractor-programming-code.html, 2017.
[4] Committee on Environment and Natural Resources, Grand Challenges for Disaster Reduction, National Science and Tech. Council, 2008.
[5] P. Berg *et al.* Order within Chaos: Towards a Deterministic Approach to Turbulence. 1984.
[6] STEM. The spatiotemporal epidemiological modeler project. http://www.eclipse.org/stem/ , 2017.
[7] C. A. Andersson and R. Bro. The N-Way Toolbox for Matlab. Chem. and Intel. Lab. Systems, 52(1):1-4, 2000.
[8] B. W. Bader, T. G. Kolda, *et al.* MATLAB Tensor Toolbox Version 2.5, Available online, January 2012.
[9] C. H. Chen, L. H. Lee, Stochastic Simulation Optimization: An Optimal Computing Budget Allocation, World scientific, 2010.
[10] R. Fisher. The Design of Experiments, 1935.
[11] R. A. Harshman, Foundations of the PARAFAC procedure: Model and conditions for an explanatory multi-mode factor analysis. UCLA Working Papers in Phonetics, 16:1-84, 1970.
[12] S. Huang, K. S. Candan, M. L. Sapino. BICP: Block-Incremental CP Decomposition with Update Sensitive Refinement. CIKM 2016
[13] B. Jeon, Inah Jeon, Lee Sael and U Kang SCouT: Scalable Coupled Matrix-Tensor Factorization - Algorithm and Discoveries. ICDE 2016.
[14] I. Jeon, E. Papalexakis, U. Kang, and C. Faloutsos. HaTen2: Billion-scale tensor decompositions. ICDE 2015
[15] N.L. Johnson. Sequential analysis: a survey. Journal of the Royal Statistical Society, Series A. Vol. 124 (3), 372411, 1961.
[16] U. Kang, *et al.* Gigatensor: scaling tensor analysis up by 100 times algorithms and discoveries. KDD 2012.
[17] M. Kim and K.S. Candan. Decomposition by normalization (DBN): leveraging approximate functional dependencies for efficient CP and tucker decompositions. Data Min. Knowl. Discov. 30(1): 1-46, 2016.
[18] S. Klus, P. Gelβ, S. Peitz, C. Schutte, Tensor-based dynamic mode decomposition, ArXiv, 2017.
[19] T. G. Kolda, J. Sun. Scalable tensor decompositions for multi-aspect data mining. ICDM 2008.
[20] T.G. Kolda and B.W. Bader. Tensor Decompositions and Applications. SIAM Rev. 51, 3, 455-500. 2009.
[21] X.Li, S.Huang, K.Candan and M.Sapino, Focusing Decomposition Accuracy by Personalizing Tensor Decomposition (PTD). CIKM'14.
[22] X. Li, S.Y. Huang, K.S. Candan and M.L. Sapino, 2PCP: Two-phase CP decomposition for billion-scale dense tensors. ICDE 2016.
[23] Sicong Liu, *et al.* Notes2: Networks-of-traces for epidemic spread simulations. AAAI Workshop on Computational Sustainability, 2015.
[24] E. Papalexakis, C. Faloutsos, and N. Sidiropoulos. Parcube: Sparse parallelizable tensor decompositions. PKDD,2012.
[25] G. Pedrielli, *et al.* Single-run Simulation Optimization Through Time Dilation and Optimal Computing Budget Allocation. Stochastic Models of Manufacturing and Service Operation, 2015.
[26] G. Pedrielli, *et al.* Empirical analysis of the performance of variance estimators in sequential single-run ranking selection: the case of time dilation algorithm. WSC'16
[27] A.H. Phan and A. Cichocki, PARAFAC algorithms for large-scale problems, Neurocomputing, vol. 74, no. 11, pp. 1970-1984, 2011.
[28] S. Poccia *et al.* SIMDMS: Data Management and Analysis to Support Decision Making through Large Simulation Ensembles. EDBT 2017.
[29] M.Rogers, L. Li, S. Russell, Multilinear Dynamical Systems for Tensor Time Series. NIPS 2003.
[30] L.W.Schruben, *et al.* Simulation Optimization Using Simultaneous Replications and Event Time Dilation, WSC 1997.
[31] C. E. Tsourakakis, Mach: Fast randomized tensor decompositions. Arxiv :0909.4969, 2009
[32] L. Tucker, Some mathematical notes on three-mode factor analysis. Psychometrika, 31:279-311, 1966.